Preprint typeset in JINST style - HYPER VERSION

The Cluster Crawler Suite Technical Manual

April 21, 2014

B. Baller[a]

[a]*Fermi National Accelerator Laboratory, Batavia, IL 60510 USA*

E-mail: `baller@fnal.gov`

ABSTRACT: The main features of the Cluster Crawler suite of algorithms are presented.

Contents

1. Introduction **2**

2. CCHitFinder Algorithm **3**
- 2.1 FitNG 3
- 2.2 Crude Hits 4
- 2.3 Study Hits 4

3. Cluster Crawler Algorithm **4**
- 3.1 RunCrawler 7
- 3.2 ClusterLoop 7
- 3.3 CrawlUS, LACrawlUS 8
  - 3.3.1 Kink Angle Check 8
- 3.4 AddHit, AddLAHit 9
  - 3.4.1 Charge Similarity Requirement 9
- 3.5 FitCluster, FitClusterMid 10
- 3.6 FitClusterChg 10
- 3.7 TmpStore, TmpGet 11
- 3.8 ChkMerge 11
  - 3.8.1 End-to-End Cluster Merge 11
  - 3.8.2 ChkMerge12 12

4. DoMerge **12**
- 4.1 ChkSignal 12
- 4.2 MergeClusterHits 12

5. FindVertices **12**
- 5.1 ClusterVertex 13
- 5.2 ChkVertex 14
- 5.3 VtxClusterSplit 14
  - 5.3.1 SplitCluster 14

6. CCHitRefiner Algorithm **14**
- 6.1 SetClusterBeginEnd 16

7. Configuring **16**

8. Debugging **17**

9. Technical Notes **21**

10. Appendix **21**

1. Introduction

Many methods for reconstructing two-dimensional line-like clusters of reconstructed hits use a global metric for pattern recognition. The best example of this is the Hough line algorithm. The intent of the Cluster Crawler algorithm is to construct clusters using local information. The "Crawler" name is derived from the similarity of this technique to "gliders" in 2D cellular automata. The concept is to construct short line-like "seed" clusters of proximate hits in an area of low hit density where hit proximity is a good indication that the hits are indeed associated with each other. Additional nearby hits are attached to the cluster end if they are similar to the hits already attached to the cluster. The term "leading edge" denotes the end of the cluster to which hits are being added while crawling. Seed clusters are formed at one end of the hit collection so that crawling in only one direction is sufficient.

Clusters are formed in a 2D plane of wire number along the abscissa and time, or more precisely TDC tick, along the ordinate. Hits are reconstructed on each wire by the hit finder algorithm CCHitFinder. The relatively slow drift velocity of electrons through the wire planes, $\sim 2\text{mm}/\mu\text{s}$, results in a hit width in time of $\sim 0.6\mu s$ which is equivalent to a spatial width of $\sim$ 1mm in the main drift volume. Ionization from close particles will appear as a single "distorted hit" if their separation is $\sim$1 mm. As a result, the time of the reconstructed hit will be erroneous and may result in poor cluster reconstruction.

The second member of this suite, Cluster Crawler, uses the physics of the passage of particles through matter to identify and ignore poorly reconstructed hits. In addition to the hit time, hit charge is an important discriminant for associating a hit with a cluster. The algorithm takes account of the expected charge variation along the particle trajectory due to random dE/dx fluctuations and the large increase in dE/dx near the end of travel of stopping particles.

Ionization electrons from a particle traveling parallel to the wire plane will arrive at approximately the same time. The arrival time distribution depends only on their trajectory through the wire planes and on diffusion. Hits constructed from this charge should all have similar area and time. The charge is $\propto \sqrt{\text{Amp} \times \sigma}$. The arrival time of ionization electrons from a particle traveling at some angle, θ, with respect to the wire plane will have a wider time spread and more charge will be deposited on each wire but these hits will still be similar to those belonging to the same cluster on neighboring wires. We use the term dT/dW = "delta Time" / "delta Wire" to denote the tangent of θ (times a scale factor to convert into physical space coordinates).

The goal of the third member of the suite, CCHitRefiner, is to correct poorly reconstructed hits made by CCHitFinder using information provided by Cluster Crawler. This information includes 2D vertices as well as clusters. CCHitRefiner is under development and is not ready for general use. In principle, CCHitRefiner should enable reconstruction of very low energy particles with T < 20 MeV produced in neutrino interactions. The last function of CCHitFinder is to use the ratio of charge at the two ends of the cluster to determine it's direction.

The three algorithms are called by the producer module ClusterCrawler_module which places the final set of hits, clusters and 2D vertices in the event record.

In this report, the preceding "f" on the text fVariable identifies it as a user-defined tracking variable. Text in the `typewriter` font denote variables as they are used in the source code.

2. CCHitFinder Algorithm

Wire signals from induction and collection wire planes are converted into Gaussian-like shapes by deconvolution in the CalWire module. The CCHitFinder algorithm differs from other hit finders by the fitting method. Wire signals are fitted to one or more Gaussian distributions down to the noise threshold fMinSig = fMinSigInd for induction planes or fMinSig = fMinSigCol for the collection plane. The Gaussian fit χ^2/DOF is used to identify distorted hits - those that likely contain charge from different particles. Deconvolved hits are not necessarily Gaussian in shape nor will they be similarly (non) Gaussian in all wire planes. To minimize this problem, a vector of χ^2/DOF normalization factors, fChiNorms, is input from the fcl file. The method for determining these factors is described in the Study Hits section2.3.

CCHitFinder searches the wire signal for Regions Above Threshold, or RATs, on each wire. A RAT is a contiguous block of ticks in which the signal, S, exceeds the threshold. Hits that lie within the RAT will have a minimum time width of minRMS = fMinRMSInd (induction planes) or minRMS = fMinRMSCol (collection plane), therefore the length of a RAT is required to exceed $2\times$ minRMS. Bumps in the RAT are found by requiring $S_t > S_{t-1}$ and $S_t > S_{t+1}$ and $S_{t-1} > S_{t-2}$ and $S_{t+1} > S_{t+2}$, where S_t is maximum amplitude of the bump and t is the time of the maximum amplitude. A "Crude" hit, described in section 2.2, is created if the number of bumps exceeds fMaxBumps. The bump times are put into to the `bumps` vector and passed to the fitting routine, FitNG.

If the normalized χ^2/DOF of the fit exceeds fChiSplit it is presumed that more hits exist in the RAT than there are bumps. Additional fits are performed with additional "hidden" bumps, until the fit χ^2/DOF is less than fChiSplit or until fMaxXtraHits are included in the fit.

The reconstructed hits are stored in a temporary vector `allhits` of `CCHit` structs by the routine StoreHits. This struct contains all of the information required to create a final hit that is stored in the event, although in a somewhat different form than `recob::Hit`. It also defines variables that will later be modified by ClusterCrawler and CCHitRefiner. For example, the existence of a "multiplet" of hits in one RAT, that is `allhits[].numHit` > 1, is an indicator of either closely separated tracks or a single highly-inclined track. `allhits[].LoHitID` is the index of the lowest time hit in the RAT. This information is used by ClusterCrawler to merge hits on large angle clusters as described in the next section. ClusterCrawler also sets `allhits[].InClus` to the ID of the cluster it is associated with. A value of 0 indicates that the hit is not associated with any cluster. A value < 0 indicates that the hit is obsolete.

2.1 FitNG

The RAT is fit to `nGaus` Gaussian distributions, where `nGaus` is the number of bumps found plus the number of hidden bumps. The `bumps` vector provides an initial estimate of the Gaussian fit parameters for the found bumps. The difference between the wire signal vector S and an expected signal vector S_{expect} (made under the hypothesis that there are `bumps.size()` bumps) is used to estimate the parameters of the hidden bumps.

After fitting, the bump parameters are sorted in increasing time order. Quality checks are then made to ensure that i) the fitted bump times are within the RAT boundaries, ii) the bump widths are reasonable (0.5 * minRMS < fitted RMS < 5 * minRMS), iii) the bumps are not too similar in time (δT > 2), and iv) all bump amplitudes exceed fMinSig. The fit χ^2/DOF is set large if all of these conditions are not met.

2.2 Crude Hits

This routine is called when any type of fitting failure occurs. It creates a single crude hit from the signals in a RAT. The charge-weighted mean hit time and rms are found. A crude hit amplitude is then calculated using the formula $\Sigma S_i/(\sqrt{2\pi} \times \mathrm{RMS})$.

2.3 Study Hits

The StudyHits routine provides a mechanism for optimizing the settings for a specific detector configuration. It need only be run when wire signals are expected to change, e.g. for a new detector or if the deconvolution kernel is changed. The code is commented out for normal processing. The first step prior to using this code is to identify a single real or MC event that contains a single small angle track ($dT/dW \approx 0$). The assumption is made that the hit simulation has already been adjusted such that the shape of simulated hits is the same as the shape of real detector hits. The goal of StudyHits is to determine the values of fChiNorms, fMinRMSInd and fMinRMSCol such that only one minimum-width hit is reconstructed by CCHitFinder on each wire along the selected track. Experiment-specific code modifications may be required to ignore hits that are not associated with the selected track, e.g. if an event with an isolated track cannot be found.

StudyHits is called five times while processing the event. The study vectors are initialized on the first call. The routine is called a second time after a RAT is found on a wire and a determination is made whether the RAT should be studied. The routine is called a third time after the first Gaussian fit has been done. Statistics on the fit results are accumulated as well as information to allow calculation of the cluster slope dT/dW. The number of RATs that contain a single bump are counted on the fourth call. StudyHits is called a fifth time after hit finding is completed to print out the recommended values of fChiNorms, fMinRMSInd and fMinRMSCol. These values should be entered into the fcl file and the event processed again to confirm that the newly entered values are the same as the resulting recommended values.

3. Cluster Crawler Algorithm

Pattern recognition of lines is significantly simpler when the density of hits is low. High momentum particles produced in beam neutrino interactions will likely survive well away from the interaction where the hit density is generally lower than closer to the interaction. The region near the primary interaction is troublesome for hit reconstruction because of overlapping charge depositions. The approach used here is to fit very long clusters of hits to a line far from the primary interaction vertex, downstream (DS) in the Wire-Time coordinate system, and extrapolate or crawl them towards the upstream (US) primary interaction region. Hits associated with these clusters are then removed from consideration when shorter lower-momentum clusters are reconstructed on subsequent passes through the hit collection. The mechanism for selectively tracking high vs low momentum particles is by fitting a large number of hits at the upstream end to a line on the first pass then fitting to a smaller number of hits on the next pass.

Lower momentum particles have more curvature than higher momentum particles which will result in incomplete clusters when they are reconstructed on the first pass. These are stored however if they satisfy the length requirement for the next pass. The result after tracking a low momentum particle will be an end-to-end series of clusters which have an apparent kink angle between them. A routine is optionally called after every pass to merge clusters that have this signature.

The convention used in this document is that US (DS) is in the direction of decreasing (increasing) wire number. Despite the inherent directionality of the algorithm, it should have similar efficiency for reconstructing clusters in any direction, except for those particles that travel along the drift direction.

The fcl inputs to ClusterCrawler are listed below. The values of these variables used for MicroBooNE are shown in { }'s.

fNumPass Number of passes that are made through the hit collection. {3} 3.2

vector<fMaxHitsFit> Maximum number of hits that are fitted to a line on the leading edge of a cluster. {20, 8, 4} 3.2

vector<fMinHits> Required minimum number of hits on a cluster produced on each pass. {20, 8, 3} 3.2

vector<fNHitsAve> Number of hits that should be used to define the average charge. {8, 8, 0} 3.6

vector<fChgCut> Maximum fractional hit charge difference between the average charge at the leading edge of the cluster and a nearby hit on the next US wire. {0.65, 0.65, 0.7} 3.4.1

vector<fChiCut> Maximum χ^2 for tracking a cluster. {3., 3., 3.} 3.5

vector<fMaxWirSkip> Maximum number of wires (that have high charge nearby hits) to skip over without adding a hit to the cluster. {25, 8, 3} 3.3

vector<fMinWirAfterSkip> Minimum required number of hits on consecutive wires that must be added to the cluster after skipping a high charge region. {3, 2, 1} 3.3

vector<fKinkChiRat> Monitor the fractional χ^2 increase of the last 3 cluster line fits at the US end to identify kinks. If this condition is met, the kink angle is calculated and a cut is made using fKinkAngCut. {1.2, 1.2, 0.} 3.3

vector<fKinkAngCut> Kink angle cut (radians) for cluster merging and terminating crawling. {0.35, 0.35, 0.4} 3.3

vector<fDoMerge> Boolean variable to perform cluster merging routine after tracking is finished on a pass. {false, true, true} 4

vector<fTimeDelta> Maximum time difference at the matching wire for merging clusters. {2., 3., 10.} 4

vector<fFindVertices> Boolean variable to find 2D vertices. {false, true, true} 5

fLAClusAngleCut Use Large Angle cluster tracking code if the cluster angle with respect to the wire plane exceeds this value ($< 90^\circ$). Set to 0 to turn off Large Angle cluster tracking. {45} 3.2

fClHitMergeChiCut Merge hits in a multiplet if the hit separation χ^2 is less than this value. Set < 0 to turn off hit merging. {4.} 4.2

fClGhostHitFrac Merge $\sim$parallel (Large Angle) ghost clusters if they share > this fraction of hits in a hit multiplet. Set < 0 to disable merging ghost clusters. {-0.5}
fHitErrFac Factor to specify the hit time error for fitting = fHitErrFac * `CCHit.RMS`. {0.4} 3.53.4
fAllowNoHitWire Number of allowed wires (that have no hits on them) to skip while tracking. This allows tracking to continue when the LAr purity is low. {0}
fDebugPlane Print tracking information for this plane. Set < 0 for no printing. {-1}
fDebugWire Print tracking information for clusters that start on this wire. Set < 0 to print cluster merging information. {0}
fDebugHit Print out tracking information for clusters for which the time of the most DS hit is within ± 10 ticks of fDebugHit. Set < 0 to print vertex reconstruction information. {0}

The vector variables are those which may be altered on each pass. The length of the vectors must be $\geq$ fNumPass.

ClusterCrawler reconstructs temporary clusters and temporary 2D vertices in all cryostats, TPCs and wire planes. These are stored in the vector `tcl` of `ClusterStore` structs and the vector `vtx` of `VtxStore` structs respectively. Each cluster and vertex is assigned a “CTP” code that is unique for each Cryostat, Tpc and Plane.

A cluster “Begins” at the most DS hit and “Ends” at the most US hit. The `fcl2hits` vector contains an ordered set of indices of hits on the cluster that is under construction. `fcl2hits[0]` is the most DS hit and `fcl2hits[fcl2hits.size()-1]` is the most US hit. Hits are added to the leading end of a cluster with the `push_back()` std::vector member function. Only one hit per wire may be assigned to a cluster. The following variables define the cluster under construction:

clpar Line fit parameters at the position of the US end wire, T_o and dT/dW.
clparerr Errors on the line fit parameters.
clChisq χ^2/DOF of the last fit.
fAveChg Average charge of hits on the US end of the cluster.
fChgSlp Slope from the charge fit.

These variables are updated as hits are added to the cluster. When tracking of the cluster is complete and meets the requirements for the current or next pass, it is stored in the `tcl` vectorof ClusterStore structs. ClusterStore has the following members:

ID Cluster ID (> 0).
ProcCode Code to identify the processor algorithm that created or modified the cluster.
StopCode Code to identify the reason for crawling termination.
CTP Cluster CTP code.
BeginSlp, BeginSlpErr dT/dW and error at the DS end of the cluster.
BeginTim Time of the cluster fit at the DS wire.
BeginChg Average charge of hits at the DS end.
EndSlp, EndSlpErr dT/dW and error at the US end of the cluster.
EndTim Begin time of the cluster at the US wire.
EndChg Average charge of hits at the US end.
BeginVtx Index of the vertex at the DS end of the cluster. Set < 0 if no vertex associated with this cluster.

EndVtx Index of the vertex at the US end of the cluster. Set < 0 if no vertex associated with this cluster.

2D vertices are stored in the `vtx` vector of VtxStore structs. VtxStore has the following members:

Wire Vertex wire number (in float precision).
Time Vertex time.
Wght Vertex weight.
Topo Vertex topology code.
CTP CTP code.

3.1 RunCrawler

RunCrawler loops over all cryostats, TPCs and planes. GetHitRange is called when the plane number changes to initialize the WireHitRange vector<pair> where the first (second) element of the pair is the index of the first (last) hit on each wire in the plane. A value of -2 for the first pair element indicates that there are no hits on the wire. A value of -1 indicates that the wire is dead. The 0^{th} element of WireHitRange is the first wire, fFirstWire, in the plane that contains a hit. The crawling algorithm jumps across dead wires but stops if a wire with no hit is encountered. The variable fLastWire is the the last wire in the plane that contains a hit.

RunCrawler next calculates conversion factors that are used by the crawling routines and then calls the crawling routine ClusterLoop.

3.2 ClusterLoop

The algorithm begins by searching for a short "seed" cluster consisting of three hits on three adjacent wires. The search starts at fLastWire and ends at fFirstWire + 3. The hits must have not already been included in a cluster or have been declared obsolete, i.e. `allhits[].InClus` $\neq$ 0. The charge on the most DS hit should not exceed $2\times$ the charge of the hit on the next US wire. This requirement prevents starting a new cluster using hits that likely were produced from different particles, e.g. a δ-ray close to a muon. The rationale for using the factor of two is described in the Appendix 10.

The routine VtxConstraint is called to ensure that the seed cluster does not cross a vertex. The three cluster hits are fitted to a line and a χ^2 test performed. The cluster "Begin" wire, time, slope, slope error and charge are stored in temporary variables.

Two crawling routines exist - one for small angle clusters, CrawlUS 3.3, and a variant for Large Angle clusters, LACrawlUS) 3.3. LACrawlUS is called if Large Angle cluster crawling is requested, i.e. fLAClusAngleCut > 0 and the cluster slope exceeds the specified value. The crawling routine projects the cluster under construction to the wires upstream of the leading edge, adds hits, and re-fits the cluster parameters as necessary.

After crawling is complete, the routine QACheck is called to check the cluster quality. It clears the `fcl2hits` vector if the cluster fails the requirements. The cluster is stored by the routine TmpStore if i) the number of hits is $\geq$ the minimum number of hits required for the current pass, fMinHits[pass], or ii) if the number of hits is $\geq$ the minimum number of hits required for the next pass, if pass < fNumPass - 2. The second, looser requirement provides a significant performance increase.

After all wires and hits have been considered for the current pass, two routines are optionally called. ChkMerge 3.8 merges clusters created in all completed passes if fDoMerge[pass] is true. FindVertices 5 is then called if fFindVertices[pass] is true.

After all passes through the hit collection are completed for the current wire plane, two routines analyze and possibly modify the new set of clusters and vertices that were added. MergeClusterHits 4.2 merges close hits in a multiplet; those hits whose separation is < fClHitMergeChiCut. VtxClusterSplit 5.3 looks for a vertex that lies along the trajectory of a cluster. This situation may arise when the vertex is created after the cluster is formed. If one is found, the cluster is split into two - one US of the vertex and the other DS of the vertex.

3.3 CrawlUS, LACrawlUS

The crawling routines use the `fcl2hits` vector to find the wire of the most US hit on the cluster, the `lastwire`. This routine iterates over all wires, `nextwire`, between the US end of the cluster, `lastwire - 1`, and the US end of the hit collection, fFirstWire.

Crawling stops if CrawlVtxChk finds a vertex at the projected time of the cluster on `nextwire`.

A hit may be added to the cluster by AddHit 3.4. AddHit sets the state of two Boolean variables. `HitOK` is set true if a hit was added. `SigOK` is set true if there is a hit signal at the projected time of the cluster. Crawling continues if there is a hit signal even though no hit may have been added to the cluster. This feature allows crawling through (and ignoring) a region where the charge of the hits is much higher than the average charge of hits on the cluster. This situation commonly occurs when tracking a muon through a δ-ray shower. Hits in this region include charge deposited by δ-ray electrons and the muon and will likely be displaced from the muon trajectory.

If no hit was added to the cluster but there is a hit signal, the number of wires that have been skipped without adding a hit are counted. Crawling terminates if this count exceeds fMaxWirSkip [pass]. Crawling also terminates if the fit χ^2 indicates that a failure occurred in AddHit. The just-added hit is removed from the cluster, the cluster is re-fit and control is returned to CrawlUS.

If a hit was added and the fit χ^2/DOF is acceptable, the charge at the leading edge of the cluster is updated by FitClusterChg 3.6. The cluster is next checked for an abrupt change in direction, a kink, as described in the next section.

The Large Angle version of this code, LACrawlUS, differs from CrawlUS in some respects. Since hits on large angle clusters are usually a part of a hit multiplet as described in section 2 the requirement of hit charge similarity is only made after all of the hits in the multiplet are merged into one hit. A second difference is that a hit must exist on all wires of a large angle cluster - no wire skipping is allowed.

3.3.1 Kink Angle Check

The kink angle check distinguishes between normal trajectory deviations due to multiple scattering and deviations due to a hard scatter or secondary interaction. The kink check is performed in two stages to minimize the number of calls to the trigonometric function `atan()` that is required to calculate the kink angle. The first stage uses the stored values of the χ^2 of all cluster fits that were stored in the `chifits` vector by AddHit. The last entry made to the `chifits` vector is the

χ^2/DOF of the fit when the hit on `nextwire` was added to `fcl2hits`. The first evidence for a kink is a large systematic increase in the fit χ^2/DOF as the last few hits are added to the cluster. Track wandering due to multiple scattering creates a systematic increase in χ^2/DOF as well but of lesser magnitude. The variable fKinkChiRat[pass] is the first stage cut on the χ^2/DOF increase. The first stage kink requirement is that the ratio of the χ^2s of successive cluster fits must exceed fKinkChiRat[pass] for the last 3 fits. If this condition is met, the angle of the line formed by the last 3 hits is compared with the line formed by the previous 3 hits. If the angle difference exceeds fKinkAngCut[pass], the last 3 hits are removed from the cluster, the cluster is re-fit and control is returned to ClusterLoop.

3.4 AddHit, AddLAHit

AddHit attempts to add a hit on a wire, `kwire`. It sets `HitOK` true if a hit is found and sets `SigOK` true if there is a hit signal near the expected time. The cluster under construction is projected to `kwire` to find the projected time, `prtime`, of a hit on the wire and the projected time error. The rms width of the last hit added to the cluster is multiplied by an angle dependent factor and fHitErrFac. This estimated hit error is added in quadrature with the cluster projection error to find the total error, `err`. The first requirement for adding a hit is that the hit time be within ± 3 err of `prtime`, in which case `SigOK` is set true. A charge similarity cut is made next.

3.4.1 Charge Similarity Requirement

The charge of the hit is required to be similar to the charge of the previous hits added to the leading edge of the cluster, fAveChg. This requirement can only be made after fAveChg has been calculated by FitClusterChg 3.6. A value of 0 indicates that the average charge has not yet been calculated. The charge similarity requirement uses the normalized charge ratio, `chgrat` = [(`allhits[].Charge`) − fAveChg]/fAveChg[pass].

The similarity requirement is constructed to accept hits that are created by ionization processes of the parent particle and reject hits that were created by other particles, e.g. daughters. The requirement must allow for ionization fluctuations from wire to wire as well as the large increase in the deposited charge at the Bragg peak when a particle stops in the detector. A charge cut, fChgCut[pass], of ≈60% captures the majority of hits produced by the parent particle. This cut by itself will truncate the Landau distribution for minimum ionizing particles however. Figure 1 shows the dE/dx distribution of hits on through-going muons in the ArgoNeuT detector. The average dE/dx is 2.3 MeV/cm and the rms is 0.8 MeV/cm which is 34% of the average dE/dx. The low end of the distribution is at 1 MeV/cm which is 57% below the average. Setting the charge cut to 0.65 should pass all low-charge hits on muon tracks. This cut will reject muon track hits with $dE/dx > 3.8$ MeV/cm however.

This problem can be rectified by defining a higher charge ratio cut, `bigchgcut` = 1.5 * fChgCut[pass]. The hit is allowed if the following conditions are met:

- The hit charge ratio is normal for the cluster, `fabs(chgrat) < fAveChg[pass].`
- The hit charge is not too high, `chgrat` < 2.
- The hit charge is high, `chgrat < bigchgcut`, and the hit time matches the projected time within $\pm 1.5\sigma$ and the charge of the previous hit added to the cluster was not high.

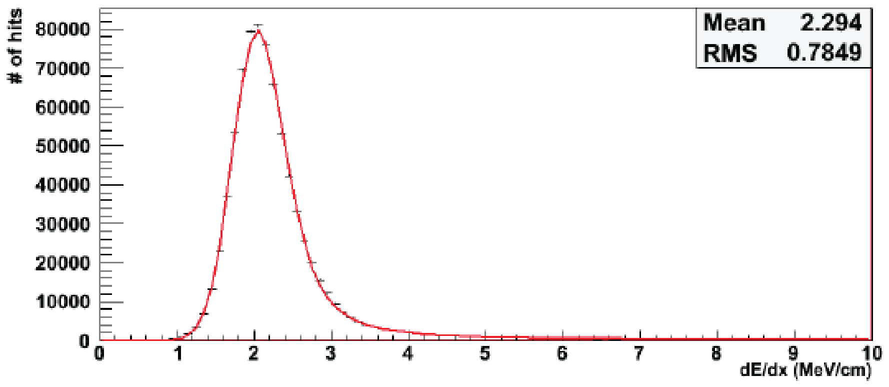

Figure 1. Distribution of dE/dx for through-going NuMI beam muons in the ArgoNeuT detector.

This cut allows high charge fluctuations on an isolated wire but rejects δ-rays whose range exceeds one wire spacing.

Note that crawling will continue to the next US wire if the hit fails these cuts since `SigOK` is set true. A similar two-stage threshold is applied to low charge hits to improve tracking efficiency if the LAr purity is low. Crawling will terminate, i.e. `SigOK` is set false, if low-charge hits are found on two consecutive wires.

After surviving all cuts, the hit is then pushed onto the `fcl2hits` vector and the cluster is re-fitted by FitCluster 3.5. The average charge at the leading edge of the cluster, fAveChg, is updated by FitClusterChg 3.6 if the newly added hit has normal charge.

3.5 FitCluster, FitClusterMid

FitCluster fits the hits on the leading edge of the cluster under construction. The hit times, T_i, and wire numbers, W_i, are fitted to a line using a weighted linear least-squares fit to two parameters - the time, `clpar[0]`, at the wire on which the US hit of the cluster resides and the slope dT/dW = `clpar[1]` where dW is the difference between the wire number of a hit assigned to the cluster and the wire number of the most US hit. A maximum of `fMaxHitsFit[pass]` hits in the `fcl2hits` vector are fitted. The hit time error is multiplied by an angle dependent factor, `angfactor = 2 - 1/(fabs(clpar[1]) )` which varies between 1 and 2.

Fitting is done by the LinFit routine adapted from Bevington[1]. The `clpar` parameters are updated if the χ^2/DOF is < 99.

FitClusterMid fits a cluster that has been stored in the `tcl` vector. It is used by routines that merge or split clusters. A linear fit is performed on a user-defined number of hits, `nhit`, starting at hit `ihtin` in the `tcl[].tclhits[ihtin]`. The hits US (DS) of `ihtin` are fitted if `nhit` is >(<) 0.

3.6 FitClusterChg

FitClusterChg calculates the average charge of hits, fAveChg, on the leading edge of the cluster under construction. The calculation method depends on the value fNHitsAve[pass]. fAveChg is set to the charge of the most US hit on the cluster if fNHitsAve[pass] = 1. It is set to the average of the two most US hits if fNHitsAve[pass] = 2. These settings are used for short clusters.

If fNHitsAve[pass] is > 2, the charge of fNHitsAve[pass] hits, Q_i, at the leading edge of the cluster are fitted to a line with intercept fAveChg at the US wire and slope fChgSlp = dQ/dW. The intent of this fit is to average the $\approx$30% variation in the deposited charge. A simple average is sufficient if there is little variation along the cluster, e.g. for minimum ionizing particles, but is inadequate for stopping particles. The calculation must also allow for stopping particles traveling either US or DS. The stopping power of stopping particles displays a power-law dependence, $dE/dx = AR^b$, near the stopping point where the residual range, R, is small. The linear relationship we use, fAveChg + $(dQ/dW) \times W$, is a crude approximation to that but it is sufficient and is computationally fast. The hit charge, Q_i, and wire number, W_i, is fitted to a line with a weighting of 50% on the charge. The value of fAveChg is updated if χ^2/DOF < 20.

3.7 TmpStore, TmpGet

The function of these utility routines is to transfer the parameters of the finished cluster into (or out of) the `tcl` vector. A cluster ID is assigned by TmpStore. A check is made to ensure that the cluster begin charge and end charge are defined. The InClus member of the `CCHit` struct is set to the cluster ID.

3.8 ChkMerge

ChkMerge attempts to merge newly created clusters. The `tcl` vector is searched for clusters having the correct CTP code for the current cryostat, TPC and plane. Clusters that were declared obsolete in prior merging operations are identified by `tcl[it1].CTP < 0` and are ignored. ChkMerge compares the “End” time, charge and angle of one cluster with the “Begin” time, charge and angle of all other clusters. The cluster merging requirements are the same as those used for crawling, i.e. fKinkAngCut[pass], fChgCut[pass], fMaxWirSkip[pass]. The two clusters may have been produced on a different pass and therefore were subjected to different requirements. ChkMerge decodes the pass for both clusters from `tcl[].ProcCode` and uses the larger pass to define the (looser) merging cuts - `angcut, chgcut, skipcut, timecut`.

The most common operation is merging clusters end-to-end. A less common situation is the presence of a short cluster that is embedded within the boundaries of a long cluster. This more complicated merge is done by a separate routine, ChkMerge12.

3.8.1 End-to-End Cluster Merge

Clusters are merged by the DoMerge 4 routine if all of the following conditions are met:

- The separation between the ends of the clusters are < `skipcut` wiress, and
- ChkSignal 4.1 finds a hit signal on all wires between the ends of the clusters, and
- the Begin/End cluster angle difference is < `angcut`, and
- the Begin/End charge ratio is < `chgcut`, and
- the Begin/End time difference is < `timecut`.

3.8.2 ChkMerge12

Cluster pairs are considered for merging if the End wire of the shorter cluster is greater than the End wire of the longer cluster and the Begin wire of the shorter cluster is less than the Begin wire of the longer cluster. The two clusters must have similar slopes and times. The indices of the clusters are passed to ChkMerge12 which compares the pattern of wires that have hits for the two clusters. A gap in the wire occupancy must exist in the long cluster that will be completely filled by the shorter cluster. If this condition is met, the longer cluster is re-fit near the two matching points by FitClusterMid 3.5. The clusters are merged by DoMerge if they have similar times, angles and charge at the matching points.

4. DoMerge

DoMerge is a utility routine that merges two clusters that reside in the `tcl` vector. The hit indices from both clusters are transferred into a temporary vector of wires `wirehit` that functions like a std::map. The hit-cluster assignment InClus is set to 0. The hit indices are then transferred in the correct wire order into the `fcl2hits` vector. The Begin cluster parameters are found by re-fitting the cluster after 4 hits have been added to `fcl2hits`. The End cluster parameters are found after all hits have been transferred. The new cluster is then added to the `tcl` vector by TmpStore. The two original clusters are declared obsolete by setting their cluster ID's negative. If a vertex exists between the two original clusters, it is declared obsolete by setting `vtx[].Wght` = -1. Lastly, the cluster-vertex assignments for the old clusters are transferred to the new cluster.

4.1 ChkSignal

ChkSignal is a utility routine that checks for the existence of a hit signal between the points $(\mathrm{Wire}_1, \mathrm{Time}_1)$ and $(\mathrm{Wire}_2, \mathrm{Time}_2)$. The Boolean variable `SigOK` is set true if there is a hit in the vicinity of the projection of the line on every wire between $wire_1$ and $wire_2$.

4.2 MergeClusterHits

In some cases, two close hits may be made by CCHitFinder even though only one should have been reconstructed. For example, there should only be one hit reconstructed per wire for a small angle highly ionizing proton. Instead, two close hits may be reconstructed if the hit fit χ^2/DOF was greater than the hit splitting χ^2/DOF cut on the first fit. In this example, one of the hits has high charge as expected for a proton and is attached to the cluster, while the other hit is just above threshold and is not attached. The hit charge is therefore incorrect. MergeClusterHits inspects hit on clusters for this signature and merges the close hit pairs.

5. FindVertices

2D vertices provide a constraint on cluster reconstruction and can in principle be used to improve cluster reconstruction near the neutrino interaction vertex. The vertex finding algorithm does not search for neutral interactions or decays. No user-adjustable cuts are provided. In addition to the vertex position (Wire, Time) the `vtx` struct includes a vertex weight and a "topology" code that can be used to debug the vertex code. The appropriate cluster-vertex pointers `tcl[].BeginVtx` and `tcl[].BeginVtx` are updated when a new vertex is created.

Clusters in the `tcl` vector that have the correct CTP code are sorted by the decreasing number of hits in the routine cl2SortByLength. FindVertices first attempts to attach the cluster to an existing vertex using the routine ClusterVertex 5.1. If that is unsuccessful, the (Wire, Time) intersection point of the two clusters is calculated. The first level requirements for considering this to be a vertex are:

Topology 1: Vertex US of both clusters

- The angle between the End of cluster 1 and the End of cluster 2 is > 0.3 radians, and
- the vertex wire is within the hit collection, $\geq$ fFirstWire, and
- the vertex wire is no more than 10 wires US of the End of both clusters, and
- the vertex time is within the readout window of the TPC.

Topology 2 (3): Vertex in-between clusters

- The angle between the Begin (End) of cluster 1 and the End (Begin) of cluster 2 is > 0.3 radians, and
- the vertex wire iis between the Begin (End) of cluster 1 and the End (Begin) of cluster 2, and
- the vertex wire is no more than 10 wires US (DS) of the ends of both clusters, and
- the vertex time is within the readout window of the TPC.

Topology 4: Vertex DS of both clusters

- The angle between the Begin of cluster 1 and the Begin of cluster 2 is > 0.3 radians, and
- the vertex wire is within the hit collection, $\leq$ fLastWire, and
- the vertex wire is no more than 10 wires DS of the Begin of both clusters, and
- the vertex time is within the readout window of the TPC.

If these conditions are met, ChkVertex 5.2 is called to compare the vertex with the set of existing vertices, to perform further quality checks and to create a new vertex. After vertex finding is completed, a check is made to ensure that both ends of one cluster are not assigned to the same vertex. This error may occur when creating vertices from very short clusters. All vertices are then re-fit by FitVtx to improve the vertex location. A weight is assigned, which is simply the number of cluster hits (< 10) on all clusters that are associated with it.

5.1 ClusterVertex

ClusterVertex attempts to attach the Begin or the End of the cluster referenced by the index `it` to an existing vertex. If the appropriate end of the cluster is within 2 wires of a vertex and the projection of the cluster time at the vertex wire is < 10 ticks, ChkSignal 4.1 is called to check for the existence of wire signals on the intervening wires. If that is successful, the cluster is associated with the vertex by setting `tcl[it].EndVtx` or `tcl[it].BeginVtx` to the vertex index and the vertex position is re-fit.

5.2 ChkVertex

ChkVertex compares the (Wire, Time) of a prospective vertex that was created with two clusters with indices = `it1` and `it2` with the list of existing vertices. If the prospective vertex is within 4 wires and 25 ticks of an existing vertex, an attempt is made to attach the clusters to the existing vertex.

If the prospective vertex is found to be well separated from existing vertices, ChkSignal is called to confirm that there are hit signals on every wire between the vertex and the appropriate ends of the two clusters. A new `VtxStore` struct is created and pushed onto the `vtx` vector.

5.3 VtxClusterSplit

VtxClusterSplit splits clusters that cross vertices. This situation will arise when a high momentum parent particle undergoes a secondary interaction, creating several daughter particles but leaving the direction of the parent largely unchanged, i.e. $\delta\theta <$ fKinkAngCut[pass]. A cluster for the parent particle will be created on the first pass before any vertices are formed from the daughter particle clusters. VtxClusterSplit splits the parent particle cluster into two new clusters and makes the proper BeginVtx and EndVtx association.

Loose proximity requirements are first made between all vertices and clusters that are not assigned to that vertex to ensure that the vertex lies within the wire and time boundaries of the cluster. The list of cluster hits, `tcl[].tclhits`, is searched to find the wire with a cluster hit that is closest to the vertex wire. The cluster is not split if one of the two clusters would have < 3 hits; instead the parent cluster is re-created with a few number of hits. Cluster splitting is done by SplitCluster 5.3.1.

5.3.1 SplitCluster

SplitCluster is passed a parent cluster index, a position in the `tclhits` vector where the split should be made, and the index of an assigned vertex. The parent cluster Begin parameters are transferred into the temporary tracking variables and the DS hit indices copied to the `fcl2hits` vector. The new daughter cluster End parameters are defined by calls to FitCluster and FitClusterChg. The ProcCode of the parent cluster is transferred to the daughter cluster which is then stored by TmpStore. The new vertex assignment is made to the daughter cluster EndVtx. The cluster BeginVtx assignment of the parent cluster is transferred to the daughter.

A similar procedure is applied to hits US of the split position. In this case, the cluster End parameters are unchanged and the cluster Begin parameters are re-fit. The ID of the parent cluster is set negative to declare it obsolete. Note that the two TmpStore calls make the correct re-assignment of hit-to-cluster.

6. CCHitRefiner Algorithm

The goal of the CCHitRefiner is to refine hits in important regions. The need for such an algorithm is apparent by inspecting the hit and cluster reconstruction near the primary vertex of a QE neutrino interaction as shown in Figure 2. The lower cluster consisting of high-charge (green colored) hits, is the proton. The upper cluster with lower-charge (yellow colored) hits, is the muon. The charge of the muon hits on wires 2032 - 2036 is dwarfed by the proton ionization. This results in several problems. The hits reconstructed on these wires have incorrect charge and time which will lead to a biased measurement of the opening angle. One way of fixing this problem is to remove the hits on wires 2032 - 2036 from the proton cluster and re-fit the End parameters. A second problem is that low energy particles from nuclear break-up will not be reconstructed.

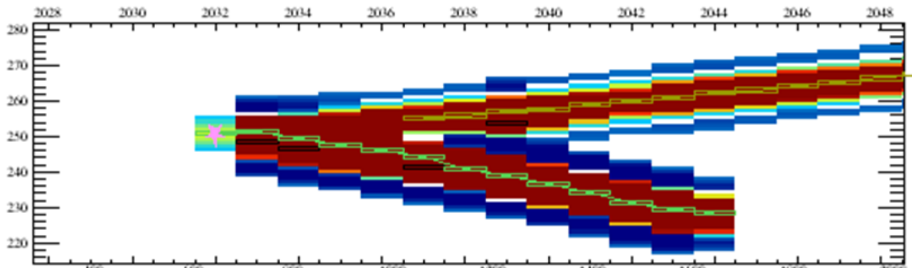

Figure 2. Hit and cluster reconstruction near a neutrino interaction vertex after processing by CCHitFinder and ClusterCrawler. The event is a QE muon neutrino interaction.

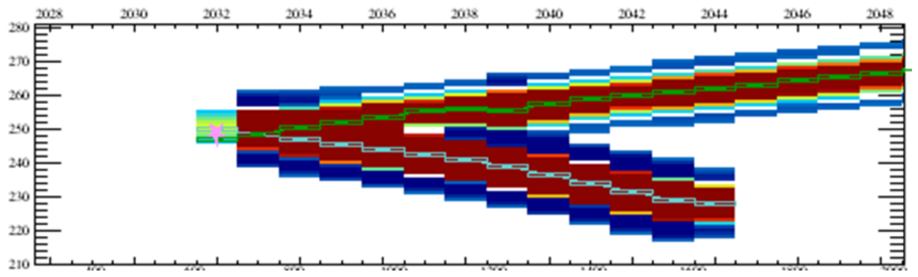

Figure 3. The same event after processing by CCHitRefiner. The reconstructed vertex wire number is 2032.96, which is erroneously drawn at wire 2032 by the event display.

A second, more elegant way of attacking this problem is to improve the hit reconstruction in the "region of confusion", by using information from well-reconstructed hits just outside the region. It is at this point that we wave our hands vigorously because the code is unfinished. The current approach is to ignore the hits within this region since their fit parameters are biased. Instead, we fit the signals on each wire in the region to different cluster - vertex hypotheses and select the best.

The main routine, RunCCHitRefiner, will loop over the vertices in each cryostat, TPC and plane (when the code is finished). The current algorithm is:

1. FindRATRange finds a range of Regions Above Threshold, a "RAT range", on each wire near the vertex that contains the region of confusion
2. FillWireSignals stores the wire signals in the RAT range in a temporary vector of vectors
3. FillVcl re-fits the clusters associated with the vertex at the boundaries of the RAT range and stores the cluster fit parameters in a temporary vector of structs, `vcl`.
4. FitVtxPos re-fits the vertex position using the `vcl` parameters.
5. FitHitamplitudes projects the time of all `vcl` clusters to each wire between the RAT boundary and the vertex. The amplitudes and times of hypothesized hit distributions for all clusters on all wires is fitted to the actual wire signals.
6. RefineHits refines the parameters of existing hits using the fit information and creates new hits if none exist.

Fitting is done using ROOT TMinuit which requires the use of two external functions and an external struct to pass data to the functions. The Minuit function fcnA is used for fitting the vertex position and average hit signals on All wires in the RAT range. The function fcnW is used to fit the amplitudes and hit times of all cluster hits on one Wire.

This rudimentary code shows some promise as one can see by comparing Figure 2 with Figure 3. Hits were created along the extrapolated trajectory of the muon on wires 2033 - 2036. The charge and charge uncertainty for these hits comes from the fitting procedure. The best fit for the vertex position is (Wire = 2032.96, Time = 248.7). One can see that the vertex is very close to the boundary between wires 2032 and 2033 by comparing the very large pulse height (red) of the hit on wire 2033 with the very low pulse height (yellow) of the hit on wire 2032.

The last routine invoked by the Cluster Crawler Suite, SetClusterBeginEnd, is described in the next section.

6.1 SetClusterBeginEnd

Many but not all particles produced in a neutrino interaction will travel DS. The cluster direction can be inferred by comparing the charge at the Begin and End of the cluster if it is significant. The user-defined parameter can be used to swap the Begin and End cluster parameters if the ratio (BeginChg / EndChg) > BEChgRat (~ 2). Begin and End vertices are then swapped and the the hit order reversed.

7. Configuring

The many input settings provide a considerable amount of flexibility which may create confusion when configuring the settings for the first time. The recommended approach is to define the settings for CCHitFinder first and only change them if one is under extreme duress. The StudyHits procedure should help with this process.

For MicroBooNE, the general philosophy is to find through-going cosmic ray muons on the first pass, find beam neutrino produced particles on the second pass and finally find low energy protons from nuclear break-up on the third pass. One or two passes may be sufficient for other detectors. Experience has shown that the algorithm is quite fast for events with little electromagnetic activity. The computational cost of adding more passes is therefore not very high. Numerous three-hit clusters will be created in events containing electromagnetic showers with energy greater than a few GeV with the MicroBooNE settings3. It is not clear that such clusters are useful in a shower reconstruction algorithm.

After configuring CCHitFinder, one should configuring ClusterCrawler with one pass configured for long clusters. The expectation for the LBNE far detector is that the very high energy cosmic ray muons will produce large δ-ray showers that will necessitate setting fMaxSkipWire[0] large. As a result fMaxHitsFit[0] must be set large so that the projection error through the δ-ray showers is small. The results can be analyzed quantitatively or qualitatively using the event display. Tracking “failures” can be traced using the method described in the next section and the input settings adjusted. A second pass should be added only after the settings for the first pass are optimized. It should be apparent that there is an element of the “butterfly effect” in the algorithm.

Changing the settings in the first pass will produce quite different behavior in subsequent passes but the resulting clusters should be very similar.

8. Debugging

Determining the cause of crawling failure can be difficult since local decisions are made on whether to include a hit in a cluster on the basis of the history of previous decisions. The first level of troubleshooting begins by producing a cluster summary report as shown in Figure 4. The report lists all clusters in the order in which they were stored. The columns show the cluster ID, CTP code, the number of hits in the cluster (nht), the Stop code, the processor (Proc) code, the Begin Wire and Time on the cluster using the format W:T, Begin angle (radians) and angle error, the Begin Charge. Similar information is displayed for the End parameters. The IDs of the BeginVtx and EndVtx associated with the cluster are in the rightmost two columns. The meaning of the Stop and Processor codes are shown in tables 1 and 2.

Stop code	Meaning
0	No hits matching the criteria were found on the next wire
1	Skipped too many occupied or dead wires
2	Failed the fMinWirAfterSkip cut
3	Stopped at a kink
4	Failed the cluster χ^2 cut
5	Cluster split by VtxClusterSplit
6	Crawling stopped at a vertex

Table 1. Stop codes.

Processor code	Meaning
N (< 10)	Produced on pass N
+ 10	Produced by cl2ChkMerge
+100	Produced by clChkMerge12
+300	Produced by LACrawUS
+1000	Modified by VtxClusterSplit
+2000	failed pass N cuts but passes N+1 pass cuts
+3000	Cluster hits were merged

Table 2. Processor codes.

A negative cluster ID indicates that the cluster is obsolete. The cluster was merged and as a result, the hits on the cluster were subsumed in a newer cluster. The new cluster that subsumed these cluster can be found by comparing the Begin wire and hit with clusters at the bottom of the report. In the example report of Figure 4, one can see that cluster 3, consisting of 63 hits with Begin W:T = 1823:8768 was subsumed by cluster 32, consisting of 157 hits which was later subsumed by cluster 56 consisting of 159 hits.

```
Clustering done in plane 0
vtx 0 wire 545 time 634 wght 26 topo 3
vtx 1 wire 991 time 8408 wght 16 topo 1
vtx 2 wire 1401 time 5848 wght 14 topo 4
  ID CTP nht Stop  Proc  beg_W:T  bTheta Therr begChg  end_W:T  eTheta Therr endChg bVx eVx
   1  0  389   0  3000  2075:9568   0.84   0.00  592  1671:7901   0.84   0.00  590  -99  -99
   2  0  255   0  3000  2017:5924   0.41   0.02    0  1755:5502   0.40   0.02  604  -99  -99
  -3  0   63   0     0  1823:8768   0.12   0.00 2219  1761:8741   0.12   0.05  437  -99  -99
   4  0  709   0  3000  1800:8480  -0.15   0.04  401  1069:8916  -0.15   0.00  374  -99  -99
  -5  0   95   0     0  1760:8741   0.12   0.00  195  1658:8696   0.12   0.03  274  -99  -99
   6  0   11   0  2000  1754:5503  -0.40   0.62  619  1744:5519  -0.41   0.08  567  -99  -99
  -7  0   34   0     0  1666:8698   0.09   0.00  310  1628:8683   0.12   0.05  265  -99  -99
  -8  0   21   0     0  1627:8681   0.12   0.00  253  1607:8673   0.12   0.00  343  -99  -99
  -9  0  111   0     0  1606:8672   0.11   0.05  709  1477:8616   0.11   0.00  273  -99  -99
  10  0    9   0  5000  1580:3213   0.24   0.00   52  1572:3204   0.33   0.05 1444  -99  -99
  11  0    9   0  5000  1571:3207  -0.28   0.42   52  1563:3221  -0.41   0.04 1120  -99  -99
  12  0  509   0  3000  1543:6090   0.93   0.01   52  1017:3410   0.94   0.00  803  -99  -99
 -13  0  103   0     0  1476:8617   0.12   0.00  202  1374:8572   0.12   0.00  312  -99  -99
 -14  0  142   0     0  1373:8571   0.11   0.00  228  1230:8509   0.12   0.00  351  -99  -99
 -15  0  229   0     0  1229:8508   0.12   0.05  714   994:8409   0.11   0.00  292  -99  -99
  16  0   13   0  5000   906:108    0.62   0.23   18   894:28     1.07   0.01 1133  -99  -99
  17  0  116   0  3000   545:633   -0.79   0.00  860   429:1075  -0.79   0.00 1098    0  -99
  18  0    3   0  2001  1822:8505  -0.47   0.00  701  1820:8509  -0.47   0.00  431  -99  -99
  19  0    5   0  2001  1802:5594   0.71   0.00  834  1798:5585   0.55   0.00  615  -99  -99
  20  0    4   0  2001  1756:8236  -0.34   0.00  696  1752:8237  -0.15   0.00  696  -99  -99
  21  0    6   0  5001  1428:5526  -0.00   0.00  331  1423:5527  -0.08   0.15 1800  -99  -99
  22  0    5   0  2001  1407:5379   0.32   0.72  756  1402:5369   0.48   0.17 1136  -99  -99
  23  0    4   0  5001  1398:5854  -0.47   0.00  655  1395:5860  -0.57   0.00  655    2  -99
  24  0    5   0  2001  1379:6199   0.78   0.00  714  1375:6193   0.41   0.35  971  -99  -99
  25  0    4   0  2001  1370:5208   0.38   0.00 1108  1367:5202   0.41   0.27 1108  -99  -99
  26  0    3   0  2001  1316:8464  -0.09   0.00  436  1314:8465  -0.09   0.00  286  -99  -99
  27  0    3   0  2001  1160:8430   0.01   0.00  393  1158:8430   0.01   0.00  219  -99  -99
  28  0    6   0  2001   999:8436  -0.08   0.00  413   994:8436   0.06   0.00  769  -99  -99
 -29  0    3   0  2001   993:8408   0.17   0.00  309   991:8407   0.17   0.00  528  -99  -99
  30  0    3   0  2001   548:643    0.66   0.08 1442   546:637    0.66   0.08 1787  -99    0
  31  0    3   0  2001   529:661   -0.65   0.06  977   527:667   -0.65   0.06 1372  -99  -99
 -32  0  157   0    11  1823:8768   0.08   0.00  216  1659:8697   0.12   0.11  429  -99  -99
 -33  0   54   0    11  1666:8698   0.11   0.00  704  1608:8674   0.15   0.00  271  -99  -99
 -34  0  213   0    11  1606:8672   0.07   0.00  259  1375:8571   0.11   0.00  308  -99  -99
 -35  0  370   0    11  1373:8571   0.05   0.00  322   996:8408   0.12   0.00  294  -99  -99
 -36  0  372   0    11  1373:8571   0.05   0.57  322   992:8407   0.09   0.11  518  -99  -99
 -37  0  266   0    11  1666:8698   0.11   0.00  704  1376:8572   0.09   0.00  282  -99  -99
  38  0  637   0  3011  1666:8698   0.11   0.45  704   993:8408   0.09   0.00  312  -99    1
  39  0    4   0     2  2041:9441  -0.39   0.00  535  2038:9446  -0.39   0.00  535  -99  -99
  40  0    3   0     2  1732:8143  -0.34   0.00 1399  1730:8146  -0.34   0.00 1021  -99  -99
 -41  0    3   1     2  1710:8721   0.22   0.00 1140  1708:8719   0.22   0.00 1394  -99  -99
  42  0    3   0     2  1674:8563  -0.20   0.00  514  1672:8565  -0.20   0.00  536  -99  -99
  43  0    4   0     2  1579:3209   0.41   0.00 1181  1576:3205   0.41   0.21 1181  -99  -99
  44  0    5   0  3002  1558:8635  -0.21   0.64  353  1554:8640  -0.25   0.00  559  -99  -99
  45  0    3   0  3002  1533:8628   0.17   0.00  407  1531:8626   0.17   0.00  519  -99  -99
 -46  0   74   0  4302  1469:4924  -1.15   0.00 1521  1396:5936  -1.36   0.01  240  -99  -99
  47  0    3   0     2  1420:5491   0.40   0.00  789  1418:5488   0.40   0.00  547  -99  -99
  48  0   28   0  3302  1401:5848  -1.31   0.00  834  1374:6204  -1.25   0.07  658    2  -99
  49  0    3   0   302  1397:5875  -1.46   0.01  354  1395:5949  -1.46   0.01  794  -99  -99
  50  0   14   4  3302  1392:5983  -1.25   0.24  824  1374:6243  -1.35   0.03  214  -99  -99
  51  0    3   0     2  1239:8799  -0.50   0.00  577  1237:8803  -0.50   0.00  698  -99  -99
  52  0    5   0   302  1157:8437  -1.16   0.17  768  1153:8466  -1.05   0.15  533  -99  -99
  53  0    6   0   302   997:8424   0.89   0.50  765   991:8407   0.55   0.00  542  -99    1
  54  0   14   0   302   533:643   -0.85   0.09 1208   517:737   -1.04   0.01 1585  -99  -99
  55  0    3   0     2   434:1076  -0.62   0.30  578   432:1082  -0.62   0.30 1088  -99  -99
  56  0  159   0  3102  1823:8768   0.07   0.00  378  1660:8697   0.14   0.00  612  -99  -99
  57  0   69   5  4302  1469:4924  -1.15   0.00 1521  1401:5854  -1.32   0.00 1760  -99    2
  58  0    5   5  4302  1400:5862  -1.39   0.03  514  1396:5936  -1.36   0.01  240    2  -99
```

Figure 4. Cluster summary report for plane 0.

The fcl inputs fDebugPlane, fDebugWire and fDebugHit trigger debugging mode. These variables serve several functions that are not apparent from their names. Debugging mode is turned off by setting fDebugPlane < 0. Setting fDebugPlane *geq* 0 prints out all clusters in plane fDebugPlane as in Figures 4. The cluster Begin (End) wire and time are printed using the nomenclature "W:T" where W is the wire number and T is the time of the first (last) hit.

To trace the cause of a crawling failure one should identify the plane, wire and approximate time of the first (most DS) hit on the cluster. Set fDebugPlane, fDebugWire and fDebugTime to these values and re-run the job. If your estimate of the hit time of the first cluster hit is within ± 10 ticks of the actual hit time, a detailed crawling report will be printed to the screen. If your time estimate was not quite correct, inspect the cluster summary report to find the closest Begin W:T pair and enter the T value into fDebugTime. Re-run the job again to see the detailed report. It is advisable to divert the screen output to a text file. The crawling decisions made at each wire and hit are printed. It is left to the user to decide how the crawling settings should be adjusted.

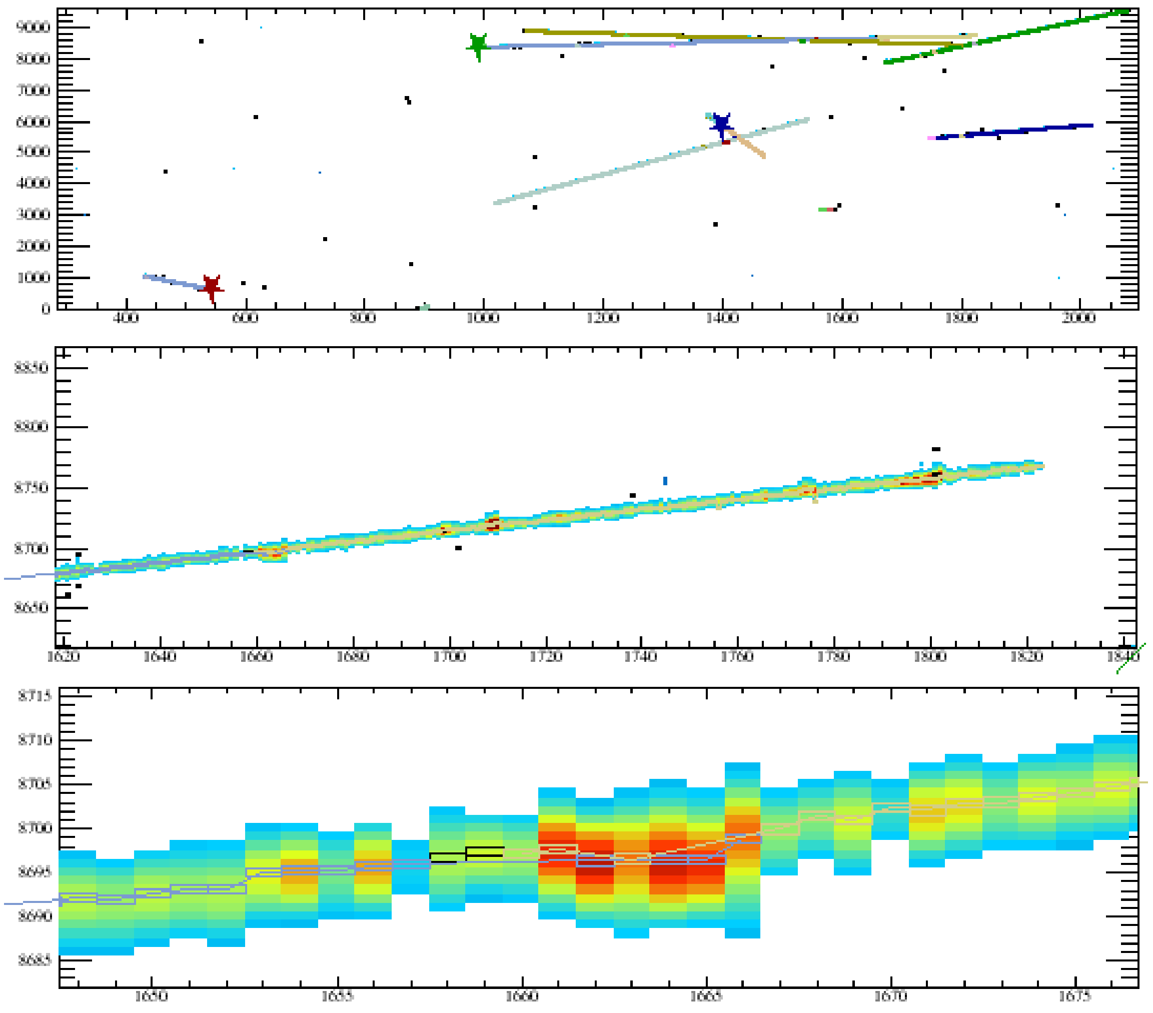

Figure 5. An event with a poorly reconstructed cluster in plane 1 as indicated by the two-color line that Begins near W:T = 1000:6200 and Ends near W:T = 800:5000 in the middle panel.

An example of a complex crawling failure is shown in the LArSoft event display in Figure 5. The top panel shows reconstructed cosmic rays in the first induction plane in the MicroBooNE detector. The color-filled stars show the location of 2d vertices. Hits associated with a cluster have the same color. The middle panel shows an enlarged view of a cluster at large drift time in the region which appears to be broken, judging from the color change near wire 1660. The bottom panel shows an extremely enlarged view in this region. The tan colored cluster entering from the right tracks through the δ-ray hits but then stops at wire 1660. Inspection of the cluster summary report shows that this is cluster 56, which after several merging operations, Begins with hit W:T = 1823:8768 and Ends on hit W:T = 1660:8697. Cluster 3 was the progenitor cluster since it has the same Begin W:T. One can tell that it was created on the first pass because the Proc code is 0 2.

We next set fDebugWire = 1823 and fDebugHit = 8768 and re-process the event again. An excerpt of the output is shown in Figure 6. The last hit added to the cluster was W:T = 1761:8741 as highlighted by the text "»ADD". Several lines below show CrawlUS looking for hits on the next wire - 1760. AddHit reports that the projected time of the cluster is between 8736 and 8746 ticks. The average charge on the leading edge of the cluster is 437. "Chk " is the output from AddHit testing each hit on the wire. The hit 1760:8741 has the correct time, but the charge is 108. The charge ratio is 75% less than the average charge and therefore fails the low charge cut. This information can be used to investigate the root cause, e.g. poor modeling of muon dE/dx fluctuations by Geant, poor algorithm in FitClusterChg for removing the fluctuations or use of an incorrect value for fChgCut[pass].

```
>>ADD W:T 1762:8742 best 0.438477 clChisq 0.131091
CrawlUS: HitOK 1 SigOK 1
Kink chk 0.131091 0.126932 0.129578 0.125921
CrawlUS: next wire 1761
AddHit: prtime Lo 8737 Hi 8746 fAveChg 437
 Chk W:T 1761:5508 InClus 2 mult 1 RMS 2.70664 Charge 679 LoTime 5502 HiTime 5514
 Chk W:T 1761:8274 InClus 1 mult 1 RMS 3.27178 Charge 752 LoTime 8267 HiTime 8283
 Chk W:T 1761:8502 InClus 0 mult 1 RMS 3.01799 Charge 452 LoTime 8496 HiTime 8509
 Chk W:T 1761:8741 InClus 0 mult 1 RMS 3.07681 Charge 376 LoTime 8735 HiTime 8748
 Best hit time 8741
ClusterFit W:T 1761:8741 1762:8742 1763:8743 1764:8743 1765:8744 1766:8744 1767:8744
1768:8744  ....
nht 20 fit par 8741+/-2 0.455731+/-0.188722 clChisq 0.0952881
 >>ADD W:T 1761:8741 best 0.285156 clChisq 0.0952881
CrawlUS: HitOK 1 SigOK 1
Kink chk 0.0952881 0.131091 0.126932 0.129578
CrawlUS: next wire 1760
AddHit: prtime Lo 8736 Hi 8746 fAveChg 437
 Chk W:T 1760:5506 InClus 2 mult 1 RMS 2.74664 Charge 525 LoTime 5501 HiTime 5512
 Chk W:T 1760:8270 InClus 1 mult 1 RMS 3.12321 Charge 613 LoTime 8263 HiTime 8278
 Chk W:T 1760:8503 InClus 0 mult 1 RMS 3.01422 Charge 298 LoTime 8498 HiTime 8510
 Chk W:T 1760:8741 InClus 0 mult 1 RMS 2.98173 Charge 108 LoTime 8737 HiTime 8746
 Best hit time 8741
 fails low charge cut. Stop crawling.
CrawlUS: HitOK 0 SigOK 0
No hit or signal on wire 1760
Check nAdjHit 3
CrawlUS done
```

Figure 6. Cluster trace report showing the last 2 hits added to cluster 3 in the cluster summary report.

To trace the cause of a merging failure one should set fDebugPlane to the appropriate plane and set fDebugWire < 0. To trace the cause of a vertex failure one should set fDebugPlane to the appropriate plane and set fDebugHit < 0.

9. Technical Notes

The code is written with the assumption that the wire number increases from US to DS. If the wire number order is reversed in the detector, tracking will be reversed as well - away from the interaction vertex instead of towards it. The tracking efficiency may be somewhat reduced as a result.

The code is written with the assumption that hits are naturally ordered by increasing wire number and increasing time.

10. Appendix

One needs to consider how the charge ratio requirement will affect the efficiency of finding seed clusters for stopping particles. A stopping particle deposits significant energy in the last mm of travel. If the particle is traveling DS, the charge ratio cut may prevent inclusion of the first, most DS, hit in the cluster. To evaluate this, we track idealized stopping protons with kinetic energy in the range 25.7 MeV < T < 30.38 MeV. Tracking is done in an Excel spreadsheet, available from MicroBooNE document data base #2975. These initial energies were selected so that the proton range varies between 9.09 mm and 12.00 mm - which is within the fourth and last wire cell of a TPC with 3 mm wire spacing. When T = 25.7 MeV, the proton traverses 3% of the last wire cell and has a stopping power of 160 MeV/cm when it enters the fourth cell but the energy deposited in that cell is only 0.5 MeV. A proton traveling 99% of the way through the fourth wire cell enters with a stopping power of 38 MeV/cm and deposits 11 MeV of energy. In Figure 7, we compare the ratio of the energy deposited in the last two cells, E_{dep4}/E_{dep3}. The charge ratio, Q_{dep4}/Q_{dep3}, will have a somewhat smaller slope due to the effects of recombination. On the other hand, the charge ratio will be sensitive to ionization fluctuations. It is for these reasons that the charge ratio for the first two hits on a seed cluster be less than two.

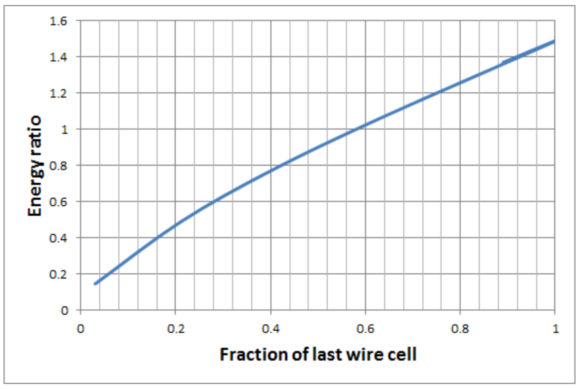

Figure 7. Ratio of the energy deposited on the last wire to the energy deposited on the next US wire for a proton traveling DS vs the fraction of the distance traveled by the proton in the last wire cell. The wire spacing is 3 mm.

References

[1] P. Bevington, 1969, *Data Reduction and Error Analysis for the Physical Sciences*, **McGraw-Hill.**